Gaussian Pulse

Amrit Singh & Nived Nandakumar

Period 7 Hannum

1 Introduction

In this lab, we simulated Schrödinger time evolution for a driven two-level system under Gaussian-shaped control pulses. Because real pulses are not perfectly identical, we compared an ideal pulse sequence to a noisy (jittered) sequence where the pulse amplitude and width vary slightly from pulse to pulse. We then quantified how much the noisy state deviates from the ideal trajectory using trace distance, which corresponds to the maximum possible difference in measurement outcomes for a single-qubit measurement.

2 Method

We modeled the qubit using a time-dependent Hamiltonian of the form

$$\hat{H}(t) = \frac{1}{2}\,\Omega(t)\,\sigma_k,$$

where the axis $k \in \{x, y, z\}$ was randomly selected for each pulse. The Gaussian pulse envelope was

$$\Omega(t) = \Omega_0 \exp\left(-\frac{(t-t_0)^2}{2\sigma^2}\right).$$

To represent realistic control noise, we introduced small statistical jitter into both Ω_0 and σ for the noisy run. After each pulse, we compared the ideal and noisy final states by forming Bloch vectors $\vec{B} = (\langle\sigma_x\rangle, \langle\sigma_y\rangle, \langle\sigma_z\rangle)$ and computing the trace distance (for pure states)

$$D = \frac{1}{2}\left\|\vec{B}_{\text{ideal}} - \vec{B}_{\text{noisy}}\right\|.$$

3 Results

Figure 1 shows $\langle\sigma_z\rangle$ vs. time for both the ideal and noisy pulse sequences. Figure 2 shows trace distance vs. pulse number.

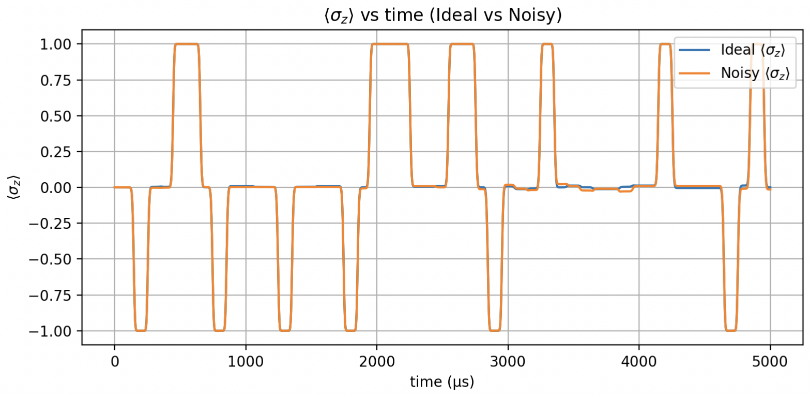

Figure 1: Expectation value $\langle \sigma_z \rangle$ vs. time for ideal and noisy pulse sequences.

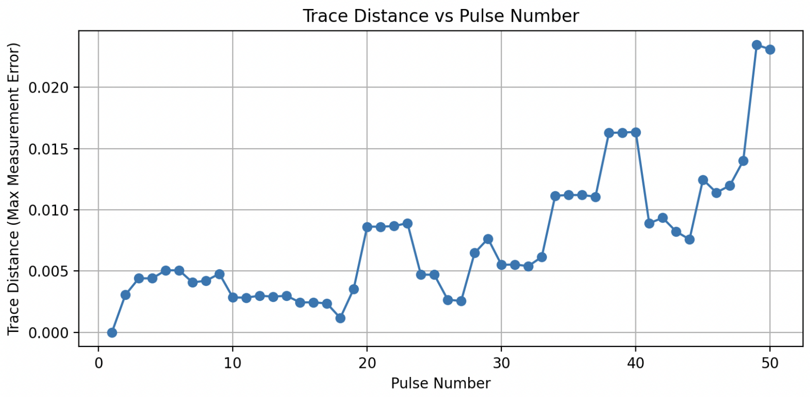

Figure 2: Trace distance (maximum measurement error) vs. pulse number.

4 Discussion (Part 3 Questions)

4.1 Ideal vs. noisy $\langle \sigma_z \rangle$

Because each pulse rotates about a randomly selected axis, the noisy curve does not show a single consistent "over-rotation" or "under-rotation" signature. Jitter sometimes increases the effective rotation angle and sometimes decreases it, so the main visible effect is gradual loss of alignment (desynchronization) between the ideal and noisy trajectories. Overall, the separation is mostly progressive rather than a one-time jump, since each pulse adds only a small deviation and those deviations accumulate over many pulses.

4.2 Trace distance behavior

The trace distance increases in an irregular way, even though it generally trends upward. This is expected because the total deviation depends on both random jitter draws and the random rotation axis at each pulse. The trace distance can also decrease at some pulses: even with jitter the evolution remains unitary, so later rotations can partially cancel earlier errors and move the noisy state closer to the ideal state again.

4.3 Sensitivity earlier vs. later

The system appears more sensitive later in the sequence. While each pulse has similar-sized jitter, errors introduced earlier propagate through all subsequent operations, so the total deviation becomes more noticeable as pulse number increases.

4.4 Connection to quantum circuits

In hardware, quantum gates are implemented using physical control pulses. If a pulse has amplitude or timing jitter, the corresponding gate deviates slightly from the intended unitary. A full circuit applies many gates in sequence, so small gate imperfections compound and cause the final state to drift away from the ideal output. The growth of trace distance represents the growing maximum measurement mismatch between ideal and noisy computation, which limits how deep a circuit can be run reliably without error correction.

A Simulation Code

The following Python code was used to generate the results and figures shown in this report. It implements sequential Gaussian pulses applied about randomly chosen axes, includes statistical jitter in pulse amplitude and width, and computes the trace distance between ideal and noisy evolutions.

```
import numpy as np
import matplotlib.pyplot as plt
import random

from qutip import (
    expect, basis, sigmax, sigmay, sigmaz,
    sesolve, Options
)

seed = 31
random.seed(seed)
np.random.seed(seed)
```

```
Omega0 = 2 * np.pi * 10e3 # rad/s
sigma_us = 10.0 # microseconds
sigma = sigma_us * 1e-6 # seconds

sigma_jitter_us = sigma_us * 0.005
sigma_jitter = sigma_jitter_us * 1e-6

Omega0_jitter = Omega0 * 0.001
t0 = 5 * sigma
t_start = 0.0
t_end = 10 * sigma

Nt = 60
tlist = np.linspace(t_start, t_end, Nt)
NumPulse = 50

sx = sigmax()
sy = sigmay()
sz = sigmaz()

opts = Options(store_states=True)

def Omega(t, args):
    return args["Omega0"] * np.exp(
        -(t - args["t0"])**2 / (2 * args["sigma"]**2)
    )

basislist = ["x", "y", "z"]

def getbasis(basislist):
    choice = random.choice(basislist)
    if choice == "x":
        return 0.5 * sx
    if choice == "y":
        return 0.5 * sy
    return 0.5 * sz

def draw_jittered_args(args_nominal):
    Omega0_draw = args_nominal["Omega0"] + np.random.normal(0, Omega0_jitter)
    sigma_draw  = abs(args_nominal["sigma"] + np.random.normal(0, sigma_jitter))
    return {"Omega0": Omega0_draw, "sigma": sigma_draw, "t0": args_nominal["t0"]}

def concat_data(result, times_all, expect_all):
    times_k = np.array(result.times)
    if times_all is None:
        times_all = times_k
        expect_all = [np.array(e) for e in result.expect]
    else:
        t_offset = times_all[-1]
        times_all = np.concatenate([times_all, times_k[1:] + t_offset])
        for j in range(len(expect_all)):
            expect_all[j] = np.concatenate([expect_all[j], result.expect[j][1:]])
    return times_all, expect_all

def getBlochVector(state):
    return np.array([
        np.real(expect(sx, state)),
        np.real(expect(sy, state)),
        np.real(expect(sz, state))
    ])

def trace_distance(b1, b2):
    return 0.5 * np.linalg.norm(b1 - b2)

psi0 = (basis(2,0) + basis(2,1)).unit()
psi_ideal = psi0
psi_noisy = psi0

args_nominal = {"Omega0": Omega0, "sigma": sigma, "t0": t0}

pulseNum = []
traceD = []

times_all_ideal = None
times_all_noisy = None
expect_all_ideal = None
expect_all_noisy = None

e_ops = [sz]

for k in range(NumPulse):
    pulseNum.append(k+1)
    H_axis = getbasis(basislist)

    H = [[H_axis, Omega]]
    args_jitter = draw_jittered_args(args_nominal)

    res_ideal = sesolve(H, psi_ideal, tlist, e_ops=e_ops, args=args_nominal)
    res_noisy = sesolve(H, psi_noisy, tlist, e_ops=e_ops, args=args_jitter)

    times_all_ideal, expect_all_ideal = concat_data(
        res_ideal, times_all_ideal, expect_all_ideal
    )
    times_all_noisy, expect_all_noisy = concat_data(
        res_noisy, times_all_noisy, expect_all_noisy
    )

    psi_ideal = res_ideal.states[-1]
    psi_noisy = res_noisy.states[-1]

    traceD.append(trace_distance(
        getBlochVector(psi_ideal),
        getBlochVector(psi_noisy)
    ))

times_us = times_all_ideal * 1e6

plt.figure()
plt.plot(times_us, expect_all_ideal[0], label="Ideal")
plt.plot(times_us, expect_all_noisy[0], label="Noisy")
plt.xlabel("time (µs)")
plt.ylabel("<z>")
plt.legend()
plt.grid()

plt.figure()
plt.plot(pulseNum, traceD, marker="o")
plt.xlabel("Pulse Number")
plt.ylabel("Trace Distance")
plt.grid()
plt.show()
```